Ledesma Ayala Angel Yeremi

HTTP

Evolución de HTTP a HTTP/2:

El Protocolo de Transferencia de Hipertexto, conocido como HTTP (por sus siglas en inglés, Hypertext Transfer Protocol), ha experimentado varias versiones a lo largo de su evolución para adaptarse a las necesidades cambiantes de la web y mejorar la eficiencia en la comunicación entre clientes y servidores. A continuación, se explora cómo ha evolucionado el protocolo HTTP desde sus inicios hasta la introducción de HTTP/2, destacando las motivaciones detrás de esta evolución:

1. HTTP/0.9 (1991): La primera versión de HTTP, HTTP/0.9, fue diseñada por Tim Berners-Lee en el CERN. Era una versión muy simple que solo permitía la recuperación de documentos HTML mediante el método GET. No incluía encabezados HTTP ni otros métodos de solicitud.

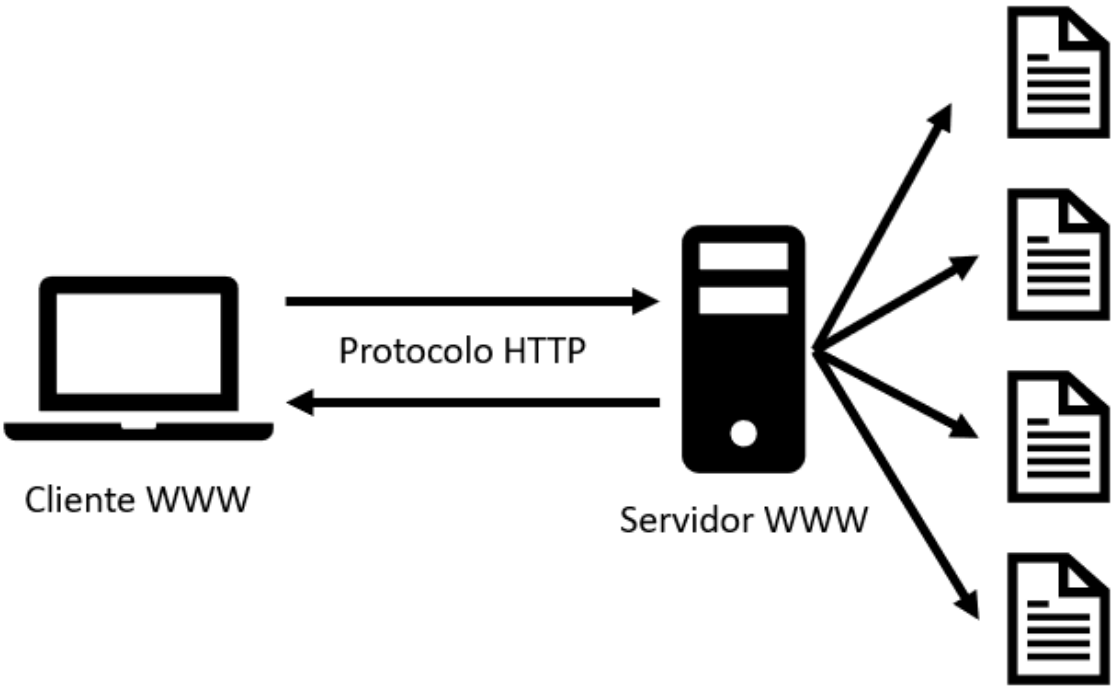
2. HTTP/1.0 (1996): HTTP/1.0 introdujo mejoras significativas al protocolo. Se agregaron nuevos métodos de solicitud, como POST, HEAD y más. También se incluyeron encabezados HTTP para permitir la negociación de contenido y la comunicación entre el cliente y el servidor. Sin embargo, HTTP/1.0 tenía algunas limitaciones, como la apertura y cierre de una conexión para cada solicitud, lo que resultaba en un alto costo de rendimiento.

3. HTTP/1.1 (1997): HTTP/1.1 abordó las limitaciones de su predecesor. Introdujo la persistencia de conexión, lo que permitía reutilizar una conexión para varias solicitudes y respuestas, reduciendo la latencia. Además, se agregó la compresión de contenido, encabezados condicionales y otras mejoras que contribuyeron a una mejor eficiencia en la transferencia de datos.

4. SPDY (2009): Aunque no es una versión oficial de HTTP, SPDY, desarrollado por Google, influyó en gran medida en la evolución de HTTP hacia HTTP/2. SPDY introdujo la multiplexación de streams, la compresión de encabezados y la gestión de flujo, entre otras características, para acelerar la carga de páginas web y reducir la latencia.

5. HTTP/2 (2015): HTTP/2 fue diseñado como una evolución de HTTP/1.1 basándose en las lecciones aprendidas de SPDY y otras tecnologías. Introdujo características clave como la multiplexación de streams, la compresión de encabezados y la gestión de flujo y priorización. Estas mejoras se centraron en hacer que la transferencia de datos fuera más eficiente y reducir el tiempo de carga de las páginas web.

Ledesma Ayala Angel Yeremi

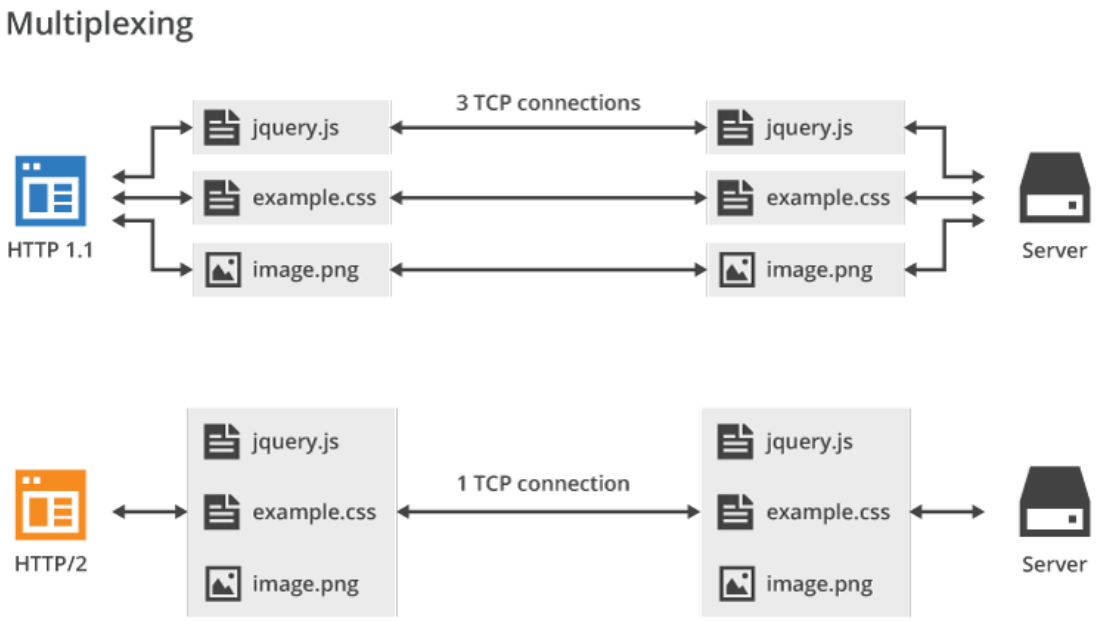


Fundamentos y Características de HTTP/2:

HTTP/2 es la segunda versión principal del Protocolo de Transferencia de Hipertexto (HTTP) y fue diseñada para mejorar la eficiencia en la comunicación entre clientes y servidores web. A continuación, se describen las características clave de HTTP/2:

1. Multiplexación de Streams: Una de las características más destacadas de HTTP/2 es la multiplexación de streams. Permite que múltiples solicitudes y respuestas se envíen y reciban simultáneamente en una única conexión TCP. Esto elimina la necesidad de abrir múltiples conexiones para cargar recursos, como imágenes, hojas de estilo y scripts, lo que reduce significativamente la latencia y mejora el rendimiento de las páginas web.
2. Compresión de Encabezados: HTTP/2 utiliza la compresión de encabezados, lo que significa que los encabezados HTTP enviados entre el cliente y el servidor se comprimen antes de transmitirse. Esto reduce la cantidad de datos redundantes y mejora la eficiencia en el uso del ancho de banda. La compresión de encabezados es especialmente beneficiosa en sitios web con muchos recursos o solicitudes.
3. Priorización de Streams: HTTP/2 permite establecer prioridades en los streams de datos para indicar cuáles son más importantes y deben recibir un tratamiento preferencial. Esto es útil en páginas web complejas con múltiples recursos, ya que permite que elementos críticos, como el contenido principal de una página, se carguen antes.

4. Gestión de Flujo: HTTP/2 introduce un mecanismo de gestión de flujo para evitar la congestión en la red. Cada stream tiene un peso y un límite de flujo que controla la cantidad de datos que se pueden enviar a través de ese stream. Esto garantiza una distribución equitativa de recursos y una experiencia de usuario más fluida.
5. Encabezados Binarios: A diferencia de HTTP/1.1, donde los encabezados se enviaban en formato de texto legible, en HTTP/2, los encabezados se envían en formato binario, lo que los hace más eficientes de procesar tanto para el cliente como para el servidor.
6. Backward Compatibility: HTTP/2 está diseñado para ser compatible con versiones anteriores de HTTP. Esto significa que los clientes y servidores que no admiten HTTP/2 aún pueden comunicarse utilizando HTTP/1.1 si es necesario. Esta compatibilidad facilita la transición hacia HTTP/2.
7. Seguridad Mejorada: Si bien no es una característica inherente de HTTP/2, se alienta el uso de conexiones seguras (HTTPS) con HTTP/2 debido a los beneficios adicionales de seguridad y rendimiento que ofrece.



Métodos HTTP y su Uso:

Los métodos HTTP, a menudo llamados "verbos HTTP", son indicadores que se utilizan en las solicitudes HTTP para especificar la acción que se debe realizar en un recurso en el servidor. Cada método tiene un propósito específico y se utiliza en diferentes contextos de la programación web y las APIs. Aquí se examinan los métodos HTTP estándar más comunes y su uso:

Ledesma Ayala Angel Yeremi

1. GET:
    - Propósito: El método GET se utiliza para solicitar datos de un recurso especificado. No tiene un efecto secundario en el servidor y se utiliza principalmente para recuperar información.
    - Uso Común: GET se utiliza en las solicitudes de navegación web cuando se accede a una página web o se solicita un recurso, como una imagen o un archivo CSS. También se utiliza en API REST para recuperar datos de recursos.
2. POST:
    - Propósito: El método POST se utiliza para enviar datos al servidor para ser procesados y, a menudo, para crear un nuevo recurso en el servidor. Puede tener efectos secundarios en el servidor.
    - Uso Común: POST se utiliza en formularios HTML para enviar datos de usuario, como al registrarse en un sitio web o enviar un comentario. También se utiliza en API REST para crear nuevos recursos.
3. PUT:
    - Propósito: El método PUT se utiliza para actualizar un recurso en el servidor o crearlo si no existe. Es idempotente, lo que significa que ejecutar la misma solicitud varias veces no tiene efectos secundarios diferentes.
    - Uso Común: PUT se utiliza en API REST para actualizar un recurso específico, como una entrada en una base de datos, utilizando los datos proporcionados en la solicitud.
4. DELETE:
    - Propósito: El método DELETE se utiliza para eliminar un recurso en el servidor. Es idempotente, lo que significa que eliminar el mismo recurso varias veces no tiene efectos secundarios diferentes.
    - Uso Común: DELETE se utiliza en API REST para eliminar un recurso específico cuando ya no es necesario.
5. PATCH:
    - Propósito: El método PATCH se utiliza para realizar modificaciones parciales en un recurso existente. Permite actualizar partes específicas del recurso sin afectar el resto de los datos.
    - Uso Común: PATCH es útil en API REST cuando se desea actualizar solo ciertos campos de un recurso sin sobrescribir todo el recurso.

6. HEAD:
    - Propósito: El método HEAD es similar a GET, pero solicita solo los encabezados de respuesta sin el cuerpo del recurso. Se utiliza para obtener información sobre un recurso sin recuperar su contenido completo.
    - Uso Común: HEAD se utiliza en situaciones donde solo se necesita información sobre el recurso, como para verificar la disponibilidad o la última modificación de un recurso.
7. OPTIONS:
    - Propósito: El método OPTIONS se utiliza para obtener información sobre las opciones de comunicación disponibles para un recurso, como los métodos HTTP admitidos o los encabezados permitidos.
    - Uso Común: OPTIONS se utiliza para habilitar la comunicación entre el cliente y el servidor y determinar qué acciones son posibles para un recurso dado.
8. CONNECT, TRACE, y otros:
    - Propósito: Estos métodos tienen usos específicos menos comunes. CONNECT se utiliza para establecer una conexión de red a través de un proxy. TRACE se utiliza para diagnosticar problemas de comunicación y suele estar deshabilitado por razones de seguridad.

Ledesma Ayala Angel Yeremi